

Angular Diciannove, Quattro.

Forms

Template Driven Forms

Reactive Forms

Kas yra forma?

HTML forma - standartinis būdas surinkti duomenis iš vartotojo.

```
<form>  
  <label for="text">Text</label>  
  <input id="text" type="text" />  
</form>
```

Kas yra form controls

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#input_types

- Form controls yra pagrindiniai formos elementai, leidžiantys vartotojui įvesti duomenis.
- Įvairūs naršyklės *native input* tipai leidžia naudoti naršyklės *native* konfigūraciją ir *input* reikšmių validaciją.

```
<form>
  <ul>
    <li>
      <label for="text">Text</label>
      <input id="text" type="text" />
    </li>
    <li>
      <label for="number">Number</label>
      <input id="number" type="number" />
    </li>
    <li>
      <label for="date">Date</label>
      <input id="date" type="date" />
    </li>
    <li>
      <label for="password">Password</label>
      <input id="password" type="password" />
    </li>
    <li>
      <label for="color">Color</label>
      <input id="color" type="color" />
    </li>
    <li>
      <label for="range">Range</label>
      <input id="range" type="range" />
    </li>
  </ul>
</form>
```

CONTACT

Name:

E-mail:

Message:

SEND YOUR MESSAGE

Contact

- Name
- E-Mail
- Message

Send Your Message

```

<form name="messageForm" action="/message" method="POST">
  <fieldset>
    <legend>Contact</legend>
    <ul>
      <li>
        <label for="name">Name</label>
        <input name="name" type="text" id="name" />
      </li>
      <li>
        <label for="email">E-Mail</label>
        <input name="email" type="email" id="email" />
      </li>
      <li>
        <label for="message">Message</label>
        <textarea name="message" id="message"></textarea>
      </li>
    </ul>
    <button>Send Your Message</button>
  </fieldset>
</form>

```

Contact

- Name
- E-Mail
- Message

Angular forms.

Template driven ir Reactive

- **Template driven forms**

- Naudoja ***two-way data-binding***, kad atspindėtų duomenų pokyčius tarp template ir kodo. Validacija atliekama naudojant **standartinius *HTML*** validavimo mechanizmus.
- Puikiai tinka paprastoms, greitoms formoms.

- **Reactive forms**

- Naudoja ***immutable*** duomenų modelį, kuris yra nepriklausomas nuo template ir kiekvieną kartą unikalus.
- Tinka **sudėtingoms, duomenų gausioms formoms**, kur reikia ***cross-field*** arba **asinchroninės** validacijos. Galima testuoti formos logiką *unit test'ais*.

<https://angular.dev/guide/forms/template-driven-forms>

Template driven forms

```
<form #expenseForm (ngSubmit)="addExpense()">
  <div class="row">
    <div class="col">
      <input
        type="text"
        class="form-control"
        placeholder="Expense name"
        name="name"
        [(ngModel)]="model.name"
      />
    </div>
  </div>
</form>
```

```
@Component({
  selector: 'app-contact',
  imports: [FormsModule],
  templateUrl: './contact.component.html',
  styleUrls: ['./contact.component.css']
})
export class ContactComponent {}
```

Template driven forms

- Importuojama iš *FormsModule*.
- Template pateikiamas formos **modelis**, kur kiekvienas form control atitinka tam tikrą modelio savybę.
- Leidžia susieti skirtingus form control, naudojant *ngForm* direktyvą.

```
@Component({  
  selector: 'app-contact',  
  imports: [FormsModule],  
  templateUrl: './contact.component.html',  
  styleUrls: ['./contact.component.css']  
})  
export class ContactComponent {}
```

```
export class Message {  
  constructor(  
    public name: string,  
    public email: string,  
    public text: string) {}  
}
```


ngModel ir two way data binding

- Naudoja ***two-way-binding*** sintaksę: reikšmei nurodyti – **[]**, o pokyčiams klausyti – **()**. Visa tai kartu sudaro ***[(ngModel)]***.
- Tai yra ***[(ngModel)]*** direktyvos ir ***(ngModelChange)*** event kombinacija.

```
<input
  type="text"
  name="message"
  [value]="message"
  (input)="message = $any($event).target.value"
/>
<input
  type="text"
  name="message"
  [(ngModel)]="message"
  (ngModelChange)="message = $event"
/>
<input type="text" name="message" [(ngModel)]="message" />
```

Template modelis, formos modelis ir template kintamieji.

- **#name** aprašo template kintamuosius Angular'e – tai kintamieji, kurie pasiekiami tik HTML template.
- Pateikia **ngForm** kaip **message** kintamąjį šablone.

```
<form #messageForm="ngForm" (ngSubmit)="sendMessage()">
```

- Pateikia **ngModel** kaip **message** variable šablone.

```
<input type="text" name="message" [(ngModel)]="message" #message="ngModel"/>
```

- „id“ naudojamas **labels**, o „name“ – form control susiejimui su forma.

Formos submit

```
<button>Send Your Message</button>
```

```
<form #messageForm="ngForm" (ngSubmit)="sendMessage()">
```

```
sendMessage() {  
    this.#service.send(this.model);  
}
```

```
<button (click)="sendMessage()">Send Your Message</button>
```

Template validacijos

- Naudoja **HTML** validacijos atributus.
- **Required, minlength, maxlength, pattern, t.t.**

```
<input
  name="name"
  type="text"
  id="name"
  [(ngModel)]="model.name"
  required
  placeholder="Your name"
/>
```

- Galima naudoti template kintamuosius, norint pasiekti validacijos parametrus:

```
<button (click)="sendMessage()" [disabled]="!messageForm.valid">
  Send Your Message
</button>
```

Template Driven State in Stiliai

State	Class if true	Class if false
The control has been visited.	ng-touched	ng-untouched
The control's value has changed.	ng-dirty	ng-pristine
The control's value is valid.	ng-valid	ng-invalid

```
.ng-invalid:not(form) {  
    border-left: 5px solid red;  
}
```

```
<label for="name">Name</label>  
<input  
  name="name"  
  type="text"  
  id="name"  
  [(ngModel)]="model.name"  
  required  
  placeholder="Your name"  
  #messageName="ngModel"  
>  
</li>  
<div [hidden]="messageName.valid || messageName.pristine">  
  @if(messageName.errors?.['required']) {  
    Message name is required  
  }  
</div>
```

Task #1 - Template Driven Forms

1. Pridėk formą prie **TasksComponent**, skirtą naujų užduočių pridėjimui. Forma turi turėti du laukus: „text“ (text input) ir „date“ (date input).
2. Naudokite šiuos įvesties laukus vietoje statinių teksto / datos reikšmių užduoties pridėjimo metu.
3. Abu laukai – „text“ ir „date“ – yra privalomi, o „date“ turi leisti pasirinkti tik datas po šiandienos.
4. Jei nėra įvestas tekstas, turi būti parodyta klaida.

<https://angular.dev/guide/forms/reactive-forms>

Reactive Forms!

```
@Component({
  selector: 'app-contact',
  imports: [ReactiveFormsModule],
  templateUrl: './contact.component.html',
  styleUrls: ['./contact.component.css'],
})
export class ContactComponent {
```

```
<form [formGroup]="messageForm"
  (submit)="sendMessage()">
  <fieldset>
    <legend>Contact</legend>
    <ul>
      <li>
        <label for="name">Name</label>
        <input
          name="name"
          type="text"
          id="name"
          placeholder="Your name"
          formControlName="name"
        />
      </li>
```

Reactive Forms

- ***Observables*** naudojami stebėti reikšmių pokyčius ir atnaujinimus.
- ***Immutable*** – kiekvieno pokyčio metu grąžinama nauja formos būseną.
- Leidžia atlikti **validaciją kode** – lengviau pritaikyti individualius sprendimus.

Reactive Forms

- Sudaryta iš **FormControl** (vienas form control), **FormGroup** (formos valdiklių map) ir **FormArray** (**FormGroup** masyvai).

```
messageForm = new FormGroup({  
  name: new FormControl(''),  
  email: new FormControl(''),  
  text: new FormControl(''),  
});
```

- Apibrėžiama kode – tik atvaizduojama šablone.

- **[formGroup]** priskiriamas *form* elementui:

```
<form [formGroup]="messageForm" (submit)="sendMessage()">
```

- **formControlName** atributas susieja HTML formos valdiklį su kodu:

```
<input  
  name="name" type="text"  
  id="name"  
  placeholder="Your name"  
  formControlName="name"  
>
```

- **FormBuilder** gali būti *inject'inamas*, kad būtų paprasčiau kurti formas:

```
messageForm =  
this.#fb.group({  
  name: '',  
  email: '',  
  text: '',  
});
```

Reactive Form Validacija

```
name: ['', [Validators.required, Validators.pattern('Jurgis')]],
```

```
get messageName() {  
  return this.messageForm.get('name');  
}
```

```
<li>  
  <label for="name">Name</label>  
  <input  
    name="name"  
    type="text"  
    id="name"  
    placeholder="Your name"  
    formControlName="name"  
  />  
</li>  
<div  
  [hidden]="messageName?.valid ||  
messageForm.controls['name'].pristine"  
>  
  @if (messageName?.errors?.['required']) {  
    <span>Expense name is required</span>  
  } @else if (messageForm.controls['name'].errors?.['pattern']) {  
    <span>Tik Jurgis</span>  
  }  
</div>
```

Template driven to Reactive

```
<li>
  <label for="name">Name</label>
  <input
    name="name"
    type="text"
    id="name"
    [(ngModel)]="model.name"
    required
    placeholder="Your name"
    #messageName="ngModel"
    pattern="Jurgis"
  />
</li>
<div [hidden]="messageName.valid || messageName.pristine">
  @if(messageName.errors?.['required']) {
    <span>Message name is required</span>
  } @else if (messageName.errors?.['pattern']) {
    <span>Tik Jurgis</span>
  }
</div>
```

Template driven to Reactive

```
<li>
  <label for="name">Name</label>
  <input
    name="name"
    type="text"
    id="name"
    placeholder="Your name"
    formControlName="name"
  />
</li>
<div
  [hidden]="messageName?.valid || messageForm.controls['name'].pristine"
>
  @if (messageName?.errors?.['required']) {
    <span>Expense name is required</span>
  } @else if (messageForm.controls['name'].errors?.['pattern']) {
    <span>Tik Jurgis</span>
  }
</div>
```

Task #2 - Reactive Forms intro

- Konvertuokite **TasksComponent** *template-driven* formą į *reactive*.
- Template validatoriai >> reactive validatoriai.

Formos statusas ir jo atnaujinimas

- **Reactive forms** pateikia būsenos atnaujinimus kaip **observables** - <https://angular.dev/api/forms/FormControl>:

```
readonly override events: Observable<ControlEvent<TValue>>;  
readonly override valueChanges: Observable<TValue>;  
readonly override statusChanges: Observable<FormControlStatus>;
```

- Taikoma **FormControl**, **FormGroup** ir **FormArray**.
- **Svarbu:** pirmasis **notification** įvyksta tik pirmo atnaujinimo metu.
- Formos pagal nutylėjimą atsinaujina **įvedant (input)**, nebent nurodyta kitaip – galima nustatyti: **input** (numatytas), **blur** – kai vartotojas išeina iš control, arba **submit** – kai pateikiama forma.

```
name: new FormControl('', {  
  updateOn: 'blur',  
  validators: [Validators.required, Validators.pattern('Jurgis')],  
}),
```

Task #3 - statuso pasikeitimai

1. Pridėkite maksimalaus simbolių skaičiaus validatorių prie „text“ įvesties lauko.
2. Stebėkite „text“ **FormControl** reikšmės pokyčius ir virš įvesties laukelio rodyk užrašą “n/(max) characters left”, kad vartotojas matytų, kiek raidžių dar gali įvesti.
3. (Optional:) Užtikrinkite, kad likusių simbolių skaičius būtų rodomas dar prieš vartotojui pradėdant rašyti.

Validacijos pagrindai

- *Validatoriai* yra funkcijos, kurios grąžina **ValidationErrors** | **null** arba **Observable<ValidationErrors | null>**.
- null reiškia, kad laukelis yra **VALIDUS**.
- **ValidationErrors** grąžina objektą, kuriame klaidos pavadinimas yra *key*, o *value* – bet koks objektas, kurį galima panaudoti klaidos pranešime.

Custom Validatoriai

- Funkcijos, atitinkančios tam tikrą *interface*, leidžia taikyti sudėtingas validacijos taisykles laukeliams.

```
import { AbstractControl, ValidationErrors } from '@angular/forms';
export const customValidator = (
  control: AbstractControl
): ValidationErrors | null => {
  const rand = control.value.startsWith('A') || Math.random();
  return rand || rand < 0.5 ? { customError: { value: control.value } } : null;
};
```

```
<div
  [hidden]="messageName?.valid || messageForm.controls['name'].pristine"
>
  @if (messageName?.errors?.['required']) {
    <span>Expense name is required</span>
  } @else if (messageForm.controls['name'].errors?.['pattern']) {
    <span>Tik Jurgis</span>
  } @else if (messageName?.errors?.['customError']) {
    <span>{{ messageName?.errors?.['customError'].value }}</span>
  }
</div>
```

Task #4 - Custom Validatoriai

1. Keiksmažodžių naudoti užduotyse nuo šiol neleidžiama.
2. Keiksmažodžiai saugomi masyve, o validatoriui reikia patikrinti, ar kuri nors iš tų reikšmių naudojama „text“ įvestyje.
3. Klaidos pranešimas turėtų būti: **„Nepageidaujama necenzūrinė leksika“**.

Cross-field validatoriai

- Panašiai kaip custom validatoriai, tačiau taikomi visam **FormGroup**, arba **FormArray**.

```
export const crossFieldValidator: ValidatorFn = (
  control: AbstractControl
): ValidationErrors | null => {
  const name = control.get('name');
  const email = control.get('email');
  return email?.value.split('@')[0] === name?.value
    ? { crossField: true }
    : null;
};
```

```
messageForm = new FormGroup({
  name: new FormControl('', {
    updateOn: 'blur',
    validators: [customValidator],
  }),
  email: new FormControl(''),
  text: new FormControl(''),
}, {validators: crossFieldValidator});
```

```
<div [hidden]="messageForm.pristine || messageForm.valid">
  @if(messageForm.errors?.['crossField']) {
    <span>Name and Email should not be identical</span>
  }
</div>
```

Async Validatoriai

- Validuoja **asincroniškai**, pvz. kviečiant validacijai nutolusį serverį.
- Funkcija, kuri priima **FormControl** reikšmę ir grąžina **Observable**.

```
interface BackendResponse {
    valid: boolean;
}
export function asynchronousValidator(httpClient: HttpClient) {
    return (ctrl: AbstractControl): Observable<ValidationErrors | null> => {
        return httpClient
            .get<BackendResponse>(`remote-url/${ctrl.value}`)
            .pipe(
                map((response: BackendResponse) =>
                    response.valid ? null : { asyncError: true }
                )
            );
    };
}
```

```
messageForm = new FormGroup({
    name: new FormControl(''),
    email: new FormControl(''),
    text: new FormControl('', {
        updateOn: 'blur',
        asyncValidators: [asynchronousValidator(this.#httpClient)],
    }),
});
```

- **SVARBU:** nustatyk **updateOn: 'blur'** validuojant, kad sumažintum užklausų į *backend* skaičių.